**CENTRE DE RECHERCHE**
**SUR LES TRANSPORTS**
(C.R.T.)
*CENTRE FOR RESEARCH*
*ON TRANSPORTATION*

UNIVERSITÉ DE MONTRÉAL
ÉCOLE DES HAUTES ÉTUDES COMMERCIALES
ÉCOLE POLYTECHNIQUE

CENTRE DE RECHERCHE SUR LES TRANSPORTS
UNIVERSITÉ DE MONTRÉAL
C.P. 6128, SUCCURSALE A,
MONTRÉAL
CANADA H3C 3J7

TÉLÉPHONE: (514) 343-7575
TÉLEX: (21) 05-561325
EMAIL: crt@crt.umontreal.ca
TÉLÉCOPIEUR (FAX): (514) 343-7121

CRT-813

# Parallel Implementations of a Branch-and-Bound Algorithm for Multicommodity Location with Balancing Requirements

by

Bernard Gendron[1]
Teodor Gabriel Crainic[1,2]

[1] Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale A, Montréal, Québec, Canada H3C 3J7

[2] Département de sciences administratives, Université du Québec à Montréal, C.P. 6192 succursale A, Montréal, Québec, Canada H3C 4R2

Centre de recherche sur les transports — Publication #813

January 1992

92-02315/

Dépôt légal – 1er trimestre 1992

**Abstract**

We present two parallel branch-and-bound algorithms for the multicommodity location problem with balancing requirements. The first algorithm implements a master-slave approach in order to perform a parallel exploration of the search tree. The second algorithm decomposes the problem in order to compute the lower bounds in parallel. We realized implementations of these algorithms on a PC-based network of Transputers. Experimental results are reported and analyzed.

Key words : Multicommodity location with balancing requirements, branch-and-bound, parallel algorithms, master-slave approach, Transputers.

**Résumé**

Nous présentons deux algorithmes parallèles de "branch-and-bound" pour résoudre le problème de localisation multiproduit avec exigences d'équilibrage. Le premier algorithme utilise une approche maître-esclave pour explorer en parallèle l'arbre d'énumération. Le deuxième algorithme permet de décomposer et de paralléliser la tâche de calcul des bornes inférieures. Ces algorithmes ont été implantés sur un réseau de Transputers. Nous présentons une analyse des résultats obtenus à l'aide de ces implantations.

Mots-clés : Problème de localisation multiproduit avec exigences d'équilibrage, "branch-and-bound", algorithmes parallèles, approche maître-esclave, Transputers.

## INTRODUCTION

The multicommodity location problem with balancing requirements typically arises in the context of the planning of the land operations of a heterogeneous fleet of containers. It is related to major logistics issues, such as the location of the depots for the empty containers and the management of empty movements.

A mathematical model for this problem has been proposed by Crainic, Dejax and Delorme (1989), who subsequently have presented a branch-and-bound algorithm to solve it (Crainic, Delorme and Dejax, 1989). The experimental results presented in this paper show that, on almost all test problems, the branch-and-bound algorithm generates a rather large enumeration tree. The results also emphasize the importance of the multicommodity aspect of the problem, since computation times increase almost linearly with the number of commodities. Hence, on almost all problems, the branch-and-bound procedure cannot determine the optimal solution in a reasonable computing time.

An attractive way to improve the performance of the algorithm is to implement it on a parallel computer. Based on the results of the sequential algorithm, we have identified two main sources of parallelism: the exploration of the search tree and the decomposition by commodity of the problem. We have developed specific algorithms to take advantage of these two opportunities for parallel computation and implemented them on a PC-based network of Transputers. The objective of this paper is to present and study the performances of these algorithms, in an attempt to determine an efficient and realistic parallelization scheme for the multicommodity location problem with balancing requirements, which may be also useful for other combinatorial problems with a multicommodity aspect.

The paper is organized as follows. In Section I, we recall the formulation of the problem and the sequential branch-and-bound algorithm. The parallel algorithms are presented in Section II. Computational experiments are reported and analyzed in Section III.

## I FORMULATION AND SEQUENTIAL ALGORITHM

The development and study of the model has been motivated by an application to the strategic/tactical planning of the land operations of an international maritime container shipping firm. A schematic description of the problem may be stated as follows. Once a ship arrives at port, the company has to deliver loaded containers, which may come in several types and sizes, to designated in-land destinations. Following their unloading by the importing customer, empty containers are moved to a depot. From there, later on, they may be delivered to customers who request containers for subsequent shipping of their own products. Furthermore, containers may be repositioned to other depots. These interdepot movements are a consequence of the regional unbalances in empty container availabilities and needs throughout the network: some areas lack containers of certain types, while others have surpluses of them.

The general problem is therefore to locate depots in order to collect the supply of empty containers available at customers' sites, and to satisfy the customer requests for empty containers, while minimizing the total operation costs: the costs of opening and operating the depots, plus the transportation costs between customers and depots and the costs generated by the interdepot movements.

The mathematical model for this problem may be formulated as follows (Crainic, Dejax and Delorme, 1989):

**Problem P1**

Minimize

$$\sum_{j\in D} f_j y_j + \sum_{p\in P}\left\{\sum_{i\in C}\sum_{j\in D}(c_{ijp}\, x_{ijp} + c_{jip}\, x_{jip}) + \sum_{j\in D}\sum_{k\in D} s_{jkp}\, w_{jkp}\right\} \tag{1}$$

Subject to

$$\sum_{j\in D} x_{ijp} = O_{ip} \qquad \forall i \in C, p \in P \tag{2a}$$

$$\sum_{j\in D} x_{jip} = D_{ip} \qquad \forall i \in C, p \in P \tag{2b}$$

$$x_{ijp} \le O_{ip} y_j \qquad \forall i \in C, j \in D, p \in P \tag{3a}$$

$$x_{jip} \le D_{ip} y_j \qquad \forall i \in C, j \in D, p \in P \tag{3b}$$

$$\sum_{i\in C} x_{ijp} + \sum_{k\in D} w_{kjp} - \sum_{i\in C} x_{jip} - \sum_{k\in D} w_{jkp} = 0 \qquad \forall j \in D, p \in P \tag{4}$$

$$x_{ijp} \ge 0 \qquad \forall i \in C, j \in D, p \in P \tag{5a}$$

$$x_{jip} \ge 0 \qquad \forall i \in C, j \in D, p \in P \tag{5b}$$

$$w_{jkp} \ge 0 \qquad \forall j, k \in D, p \in P \tag{6}$$

$$y_j \in \{0,1\} \qquad \forall j \in D \tag{7}$$

where

| | |
|---|---|
| $C$ | : set of customers; |
| $D$ | : set of candidate depots; |
| $P$ | : set of products (types of containers); |
| $f_j$ | : fixed cost of opening and operating depot $j$ ($f_j > 0$); |
| $c_{ijp}, c_{jip}$ | : unit transportation costs for product $p$ between customer $i$ and depot $j$; |
| $s_{jkp}, s_{kjp}$ | : unit transportation costs for product $p$ between depots $j$ and $k$; |
| $O_{ip}$ | : supply of product $p$ at node (customer) $i$; |

$D_{ip}$ : demand of product $p$ at node (customer) $i$;

$y_j$ : decision variable for depot $j$,

$$y_j = \begin{cases} 1, & \text{if depot } j \text{ is open} \\ 0, & \text{otherwise} \end{cases}$$

$x_{ijp}, x_{jip}$ : flows of product $p$ between customer $i$ and depot $j$;

$w_{jkp}, w_{kjp}$ : flows of product $p$ between depots $j$ and $k$.

Constraints (2) ensure that supply and demand requirements are met. Constraints (3) forbid movements through closed depots, while constraints (4) correspond to flow conservation conditions for the interdepot balancing traffic.

To solve the problem, the sequential branch-and-bound algorithm decomposes it into a potentially exponential number of subproblems. The decomposition process induces a search tree, each node being associated to a subproblem $S$, characterized by the following sets:

$D_0(S) = \{j \in D / y_j = 0\}$ : set of closed depots;
$D_1(S) = \{j \in D / y_j = 1\}$ : set of open depots;
$D_{01}(S) = \{j \in D / y_j \in \{0,1\}\}$ : set of free depots.

The subproblem $S_I$ at the root node of the tree corresponds to problem P1, i.e. in this case, all depots are free ($D_0(S_I) = D_1(S_I) = \varnothing$ and $D_{01}(S_I) = D$).

Here, follows a short description of the algorithm, called BBSEQ (see Crainic, Delorme and Dejax, 1989, for a full discussion), where

$L$ : the set of generated subproblems;
$z$ : the best upper bound identified so far.

**Procedure BBSEQ**

1. *(initialization)*
   $L \leftarrow \{S_I\}$;
   $z \leftarrow +\infty$.
2. While $(L \neq \varnothing)$ do

2.1 (*selection*)

Select a subproblem $S$ in $L$ and delete it from $L$.

2.2 (*evaluation*)

Compute a lower bound $z^{\ell}$ and an upper bound $z^{u}$ for $S$.

2.3 (*update*)

If $z^{u} < z$ then $z \leftarrow z^{u}$.

2.4 (*branching*)

If $z^{\ell} < z$ then choose $j^{*} \in D_{01}(S)$ and add to $L$ the two subproblems $S_i$ $(i = 0, 1)$ defined by the sets $D_i(S_i) = D_i(S) \cup \{j^{*}\}$, $D_{(i+1) mod 2}(S_i) = D_{(i+1) mod 2}(S)$ and $D_{01}(S_i) = D_{01}(S) - \{j^{*}\}$.

At the end of the procedure, $z$ is the optimal value of problem P1. Note also that we assume that an infeasible subproblem is characterized by $z^{\ell} = z^{u} = +\infty$.

To compute a lower bound $z^{\ell}$ at step 2.2, Crainic, Dejax and Delorme (1989) define the following linear relaxation of problem P1, called *weak relaxation*:

**Problem P2**

Minimize (1)

Subject to (2a), (2b), (4), (5a), (5b), (6) and

$$\sum_{p \in P} \sum_{i \in C} (x_{ijp} + x_{jip}) \leq \Delta \, y_j \qquad \forall j \in D \qquad (8)$$

$$y_j \geq 0 \qquad \forall j \in D \qquad (9)$$

where

$$\Delta = \sum_{p \in P} \sum_{i \in C} (O_{ip} + D_{ip}) \qquad (10)$$

It is easy to show that an optimal solution to problem P2 must satisfy constraints (8) with equality. Thus, after setting

$$y_j = \sum_{p \in P} \sum_{i \in C} (x_{ijp} + x_{jip})/\Delta \tag{11}$$

we obtain the following equivalent problem :

**Problem P3**

Minimize

$$\sum_{p \in P} \left\{ \sum_{i \in C} \sum_{j \in D} \Big( (c_{ijp} + f_j/\Delta) x_{ijp} + (c_{jip} + f_j/\Delta) x_{jip} \Big) + \sum_{j \in D} \sum_{s \in D} s_{jkp} w_{jkp} \right\} \tag{12}$$

Subject to (2a), (2b), (4), (5a), (5b), (6).

Problem P3 is a *multicommodity uncapacitated minimum cost flow problem*, which decomposes into $|P|$ independent uncapacitated minimum cost flow problems. It can then be solved by any of the known methods for this class of problems.

Using the formulation P3, it is easy to compute a lower bound $z^\ell$ for any subproblem $S$ selected at step 2.1. Define the modified depot fixed costs $f_j^{mod}$ :

$$f_j^{mod} = +\infty \qquad \forall j \in D_0(S) \tag{13a}$$

$$f_j^{mod} = 0 \qquad \forall j \in D_1(S) \tag{13b}$$

$$f_j^{mod} = f_j \qquad \forall j \in D_{01}(S) \tag{13c}$$

Then, a lower bound for $S$ is obtained by replacing in problem P3 the original fixed costs by the modified ones, and by adding to the value of this new objective function the constant term $\sum_{j \in D_1(S)} f_j$.

An upper bound $z^u$ for $S$ may also be easily computed. Let $(\overline{x}, \overline{w})$ be an optimal solution of problem P3, where the original fixed costs are replaced by the modified fixed costs, and define $\overline{y}$ in the following way:

$$\overline{y_j} = \begin{cases} 0, & \forall j \in D_0(S) \cup \{j \in D_{01}(S) / \overline{x}_{ijp} = \overline{x}_{jip} = 0, \forall i \in C, p \in P\} \\ 1, & \forall j \in D_1(S) \cup \{j \in D_{01}(S) / \exists i \in C, p \in P, \overline{x}_{ijp} > 0 \text{ or } \overline{x}_{jip} > 0\} \end{cases} \quad (14)$$

The computation of the objective function (1) at $(\overline{x}, \overline{w}, \overline{y})$ yields an upper bound for $S$.

Finally, note that the selection of a subproblem at step 2.1 is determined by the well-known depth-first rule, while the choice of $j^*$ at step 2.4 has been the subject of an extensive study by Crainic, Delorme and Dejax (1989).

## II PARALLEL ALGORITHMS

In this section, we describe two parallel algorithms based on the procedure BBSEQ, presented in the previous section. We restrict our analysis to algorithms implemented on a MIMD message-passing parallel computer, in which each processor has its own memory and there is no shared memory between processors.

There are two main sources of parallelism in the BBSEQ branch-and-bound procedure: the exploration of the search tree and the computation of a lower bound for each subproblem.

The parallel exploration of the search tree is implemented by executing steps 2.1 to 2.4 (or a subset of them) on several subproblems at the same time. This may be achieved by adopting either a *master-slave* approach, or a *collegial* approach.

In a master-slave approach, one processor, called the *master*, manages (in its memory) the list of generated subproblems. To each processor is then assigned a subproblem for which steps 2.2 to 2.4 (or a subset of them) are executed. Results are sent back to the master, which updates $L$ and $z$, and performs branching steps, if necessary.

In a collegial approach, each processor is assigned an initial subproblem, and it executes procedure BBSEQ using its own list of generated subproblems. Processors communicate upper bounds and exchange subproblems, in order to keep busy as many processors as possible.

The two approaches may be implemented by using either of two communication modes : *synchronous* or *asynchronous*. In synchronous mode, every processor stops at a predetermined point in the algorithm to send messages to, or to receive information from other processors. On the other hand, in asynchronous mode, the processors never stop, and a message can be sent or received at any time during the execution. In this case, a control mechanism must be provided to prevent multiple access to the same resource at the same time.

Our choice of an appropriate scheme for the implementation of a parallel search of the tree, is motivated by two important characteristics of the sequential algorithm, illustrated by the computational experiments. First, the time required by the evaluation step (2.2) largely dominates the time required by the other steps, even for problems with few products. Hence, it is also reasonable to suppose that evaluation time will dominate communication time. Therefore, since it has been shown that a master-slave approach may be efficient in this context (Trienekens, 1986), and that it may even dominate very sharp asynchronous collegial algorithms (Quinn, 1990), this approach seems appropriate to solve our problem.

The second characteristic of the algorithm is that, in general, it generates a rather large search tree. Indeed, for the experiments reported in the next section, the enumeration had been stopped once 1000 iterations have been performed. In this context, a desirable property of a parallel algorithm is to reach its best solution (which should be as good as the sequential one) in a number of iterations comparable to that of the sequential algorithm. To achieve this goal, a global control mechanism over the exploration is desired and, thus, a master-slave approach is to be prefered.

Finally, we note that, in the context of a master-slave approach, an asynchronous algorithm has no advantage over a synchronous one. This is due to the fact that in the asynchronous algorithm, the processors must access the same resource (namely, the memory of the master processor); hence, one gets waiting times comparable to those obtained by a synchronous algorithm. Moreover, a synchronous algorithm is easier to implement and test.

Our *synchronous master-slave* algorithm is divided into two phases: a sequential phase and a parallel phase. The sequential phase consists of procedure BBSEQ executed by the master processor up to $N$ iterations. The determination of $N$ is critical. If $N$ is too small, generally a good upper bound has not yet been found and the parallel exploration will examine subproblems not considered by the sequential algorithm. This results in useless work. If $N$

is too big, the sequential part becomes too important and the performance of the parallel algorithm decreases.

During the parallel phase, a slave processor is *active* if it executes step 2.2 (evaluation) or if it waits to send information to the master. We then define

| | | |
|---|---|---|
| $NPROC$ | : | the total number of processors available; |
| $n_p$ | : | the number of active processors; |

in order to describe the two procedures of the parallel phase: procedure MASTER, executed by the master processor, and procedure SLAVE, executed by the other processors.

**Procedure MASTER**

1. While $(L \neq \emptyset)$ do

   1.1 $n_p \leftarrow 0$.

   1.2 (*sending*)
   While $\Big((L \neq \emptyset)$ and $(n_p < (NPROC-1))\Big)$ do

   1.2.1 Perform step 2.1 of BBSEQ.

   1.2.2 Send suproblem $S$ to an inactive processor.

   1.2.3 $n_p \leftarrow n_p + 1$.

   1.3 (*computing*)
   If $(L \neq \emptyset)$ then perform steps 2.1 to 2.4 of BBSEQ.

   1.4 (*receiving*)
   For $i \leftarrow 1$ to $n_p$ do

   1.4.1 Receive the results of a subproblem $S$ from an active processor.

   1.4.2 Perform steps 2.3 and 2.4 of BBSEQ.

2. Send a termination message to all processors.

**Procedure SLAVE**

1. (*receiving*)
   Receive a subproblem $S$ or a termination message from the master processor.

2. (*computing*)
   Perform step 2.2 of BBSEQ.

3. (*sending*)
   Send the results of subproblem $S$ to the master processor.

The selection of a subproblem (step 2.1 of BBSEQ) is based on a depth criterion: the selected subproblem is among those which are deepest in the tree. We do not discriminate among subproblems with the same depth.

A second parallel algorithm may be designed by decomposing the computation of the lower bound (step 2.2 of BBSEQ), into several parts to be executed simultaneously. Since the computation of the lower bound is divided into $|P|$ independent minimum cost flow problems, it is natural to design an algorithm which aims that every processor solves the same number of problems, or that, more precisely, each processor has the "same" workload. In particular, let

$$q = |P| \text{ div } NPROC$$

$$r = |P| \text{ mod } NPROC.$$

Then, $r$ processors will each solve $(q+1)$ problems and the $(NPROC - r)$ remaining processors will each solve $q$ problems. This very natural decomposition scheme is also motivated by computational experiments, since it has been observed that the execution time increases almost linearly with the number of products.

Finally, note that the parallelization of the minimum cost flow algorithm does not fall within the scope of the present paper. On the other hand, the two preceding parallelization schemes may be combined for increased efficiency. The general idea of a combined algorithm is to implement a *hierarchy of slaves*: perform a master-slave exploration of the search tree,

while simultaneously compute lower bounds in parallel by assigning slave processors to each processor involved in the exploration.

## III COMPUTATIONAL EXPERIMENTS

The objective of the numerical experimentation is to evaluate the performance of the two parallel algorithms presented in the previous section. More precisely, we aim to answer the following questions:

1) When to begin the parallel phase of the exploration? What should be the length ($N$) of the sequential phase of the algorithm?
2) Is the depth selection criterion efficient for the parallel exploration of the tree? How does it compare with a selection criterion based on generation order?
3) How does the performance of the parallel exploration vary with the number of processors?
4) How does the performance of the parallel exploration vary with the complexity of the data set (in particular, with the number of products)?
5) How to link the processors in order to minimize communication?
6) Is the parallel computation of the lower bound efficient for all data sets?

To answer these questions, we implemented the algorithms on a PC-based network of INMOS Transputers. Each Transputer is a processor operating at 20 Megahertz, equiped with a memory of one or four Megabytes and four communication ports. The network possesses one Transputer connected to the PC, to ensure appropriate treatment of I/O operations.

Since a sequential program had already been written in FORTRAN, we used library extensions specifically developed for communication between Transputers: Parallel FORTRAN (3L, 1988) and Express FORTRAN (Parasoft, 1990).

A first version was written in Parallel FORTRAN (PF) and it was (due to software limitations) tested on a maximum of four Transputers. These Transputers were mounted on a Quadputer board (Microway, 1990a), the interconnection network making out a complete graph. In all the tests performed with this implementation, the communication speed was fixed to 10 Megabits/second.

A second implementation was done by using Express FORTRAN (EF) and it was tested on configurations of up to 16 Transputers. The Transputers were mounted on Quadputer boards linked by using a Linkputer board (Microway, 1990b). The Linkputer offers the possibility to program different interconnection network topologies. In all tests with this version, the communication speed was fixed to 20 Megabits/second.

All experiments were conducted on a set of randomly generated test problems, as well as on data based on an actual large scale application. There were 18 random problems divided into three groups of six according to the number of products (one to six). All random problems consist of (about) 25 candidate locations, 125 customers, 600 interdepot arcs and 1750 customer-depot arcs per product. The real application comprises 12 products, 87 candidate depots, 289 customers, 4197 interdepot arcs and 32733 customer-depot arcs (with a maximum of 746 interdepot arcs and 3620 customer-depot arcs per product).

Since for all test problems, the enumeration has been stopped after 1000 iterations, we have used the following measures to estimate the performance of the parallel implementations, as compared to the sequential algorithm:

- the *gap* (in percentage) between the value of the best solution found and the lower bound computed at the root node of the tree;
- the number of iterations to reach the best solution;
- the *speedup*, which is defined as the ratio between the time of the sequential algorithm on one processor and the time of the parallel implementation on $NPROC$ processors.

Hereafter, we briefly present the experiments, the main results and the general conclusions. See Gendron (1991), for a more complete set of results and further discussions.

The first test consists in experimenting with several values for the parameter $N$, to determine the appropriate moment when to begin the parallel exploration of the search tree. The tests were conducted on the random problems, by using the PF implementation on four

processors. Table I shows, for each value of $N$ tested, the number of problems for which a worse solution than the sequential one was obtained. Small values of $N$ (namely 6, 9, 12) tend to increase this number, while larger values (namely 15, 18, 21, 24) seem more appropriate. In particular, $N$=24 corresponds to a point in the enumeration when a first feasible (integral) solution is about to be identified by the lower bounding procedure. It was also noted (Gendron, 1991) that, for this value, the number of iterations required to reach the best solution compares advantageously with the number of iterations required by the sequential implementation. These observations made us adopt a very simple and robust rule: *start the parallel exploration when a first feasible solution is identified.* For the remaining experiments, we have used this rule for the real application, and $N$=24 for the random problems.

**Table I. Problems with worse solution (Length of the sequential phase)**

| $N$ | 6 | 9 | 12 | 15 | 18 | 21 | 24 |
|---|---|---|---|---|---|---|---|
| | 5 | 3 | 3 | 1 | 1 | 1 | 1 |

In the second experiment, we analyzed the worth of selecting subproblems according to the depth rule by comparing two implementations of the parallel exploration: the first incorporates the depth selection rule, while for the second, a subproblem is selected according to its order of generation (generation order rule). The tests were conducted on the random problems, by using the PF implementation on four processors, as well as the EF implementation on 16 processors connected by a binary tree topology (Figure 2). Table II shows for the two rules the number of problems for which a worse solution than the sequential one was obtained. The two implementations behave similarly on four processors, since, in this case, the generation order rule also selects subproblems in the deep portions of the tree. On 16 processors however, the generation order rule selects more unpromising subproblems, particularly at the top of the tree, which are not selected by the depth rule. Hence, the depth rule appears superior and has been used in all subsequent analyses.

**Table II. Problems with worse solution (Selection rule)**

| *NPROC* | 4 | 16 |
|---|---|---|
| Depth rule | 1 | 1 |
| Generation order rule | 0 | 9 |

The speedup factor was the object of the third experiment. To illustrate the results of this experiment, we have chosen three representative test problems: two random problems, one with one product (R1) and another with six products (R6), as well as the real application (RA). Table III shows the sequential times obtained for these problems, while Figure 1 displays the speedup curves according to the number of processors. The EF implementation was used and, in the case of the random problems, the processors were connected by a binary tree topology (Figure 2), while a 3–ary tree topology (Figure 3) was used in the case of the real application.

The results show that the speedup increases sublinearly with the number of processors. This is consistent with the results of other parallel branch-and-bound algorithms for optimization problems reported in the literature (see, for example, Pardalos and Rodgers, 1990; and Quinn, 1990). We also note that we obtain a better speedup when the random problem has a larger number of products. However, the speedups attained for problem R6 are better than that obtained for the real application which has 12 products and a bigger network. This is mainly due to the fact that the parallel phase for the real application starts later than for the random problems. Finally, the complete results of these tests (Gendron, 1991) show that the algorithm is robust, in the sense that the solutions obtained by the parallel algorithm are similar to those obtained by the sequential procedure, irrespective of the number of processors.

**Table III. Sequential time (s) for three problems**

| Problem | R1 | R6 | RA |
|---|---|---|---|
| | 684 | 4737 | 10080 |

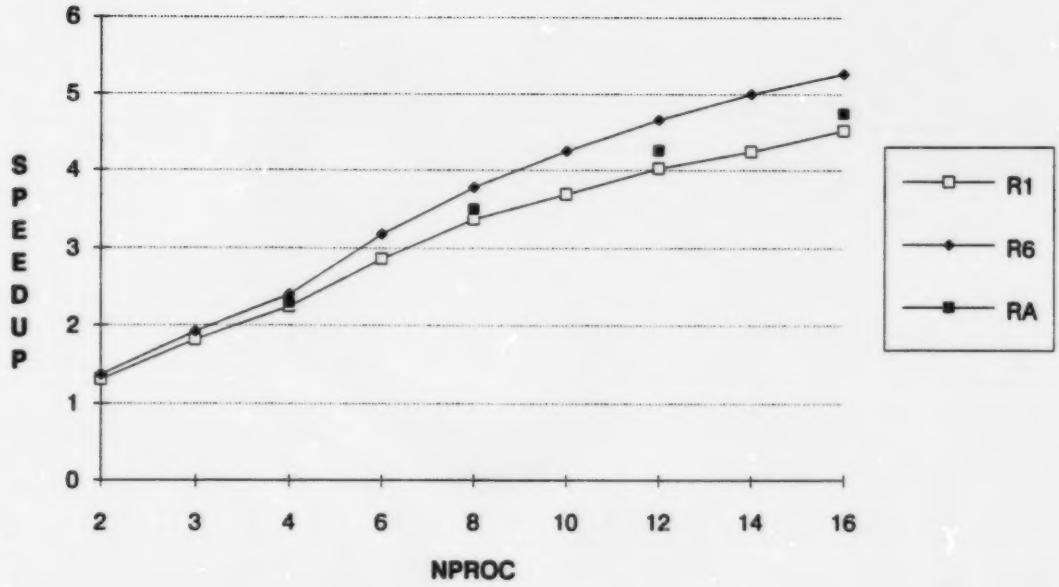


**Figure 1. Speedup curves for three problems**

Our fourth experiment consists of testing the random problems on different interconnection topologies, by using the EF implementation on 16 processors. We have tried three different topologies: the binary tree (Figure 2), the 3–ary tree (Figure 3) and the "4–3" tree (Figure 4). In all figures, the letter "H" indicates the host processor (the PC), the letter "M" indicates the master processor, while the dot lines indicate artificial links, used by Express to boot the network but never used during the execution. The objective of minimizing the communication time can be achieved by minimizing the distances between the master and the other processors. Clearly, the "4–3" tree is optimal in that sense. However, it increases the I/O time since the master is not directly connected to the host. Gendron (1991) reports that the 3–ary and "4–3" trees permit to attain comparable speedups, which are about 2% better than the speedups obtained by using a binary tree configuration. In conclusion, the 3–ary tree is to be prefered, since its structure can be easily generalized to a larger number of processors, contrary to the "4–3" tree for which the connection with the host must be changed when more processors are added (thus resulting in further increase in I/O time).

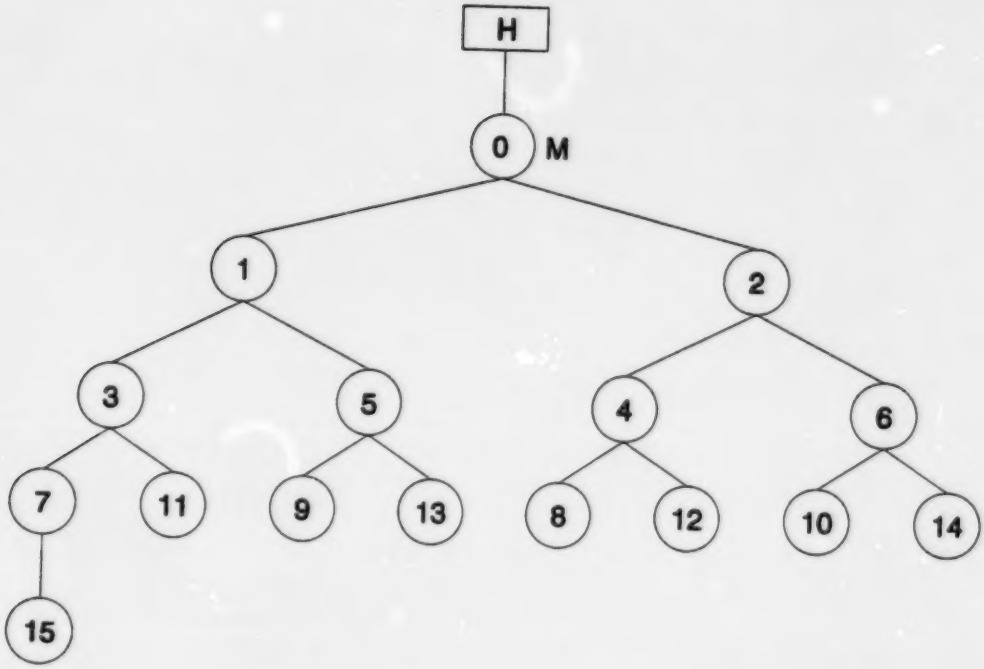


**Figure 2. Binary tree topology**

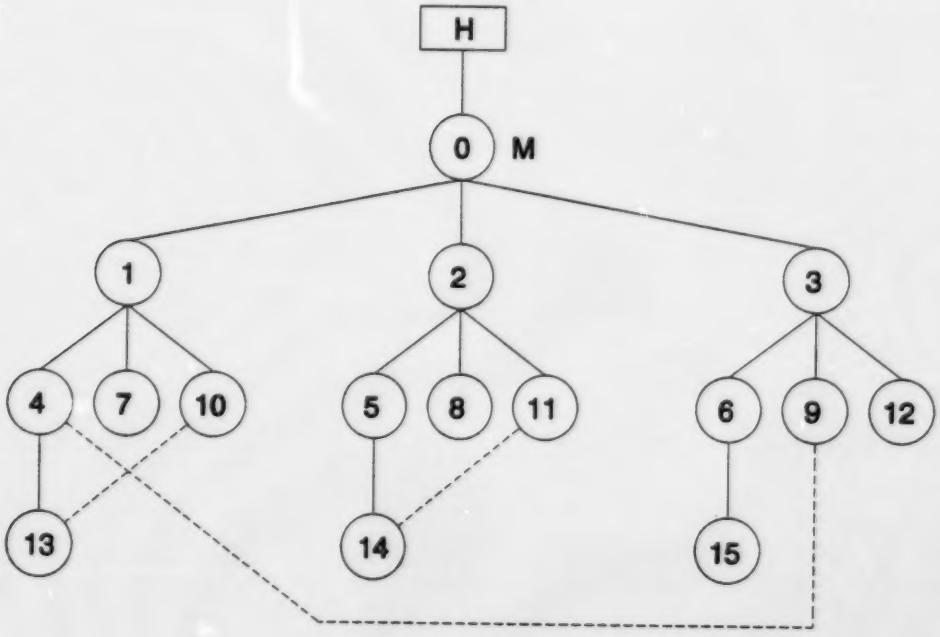


**Figure 3. 3-ary tree topology**

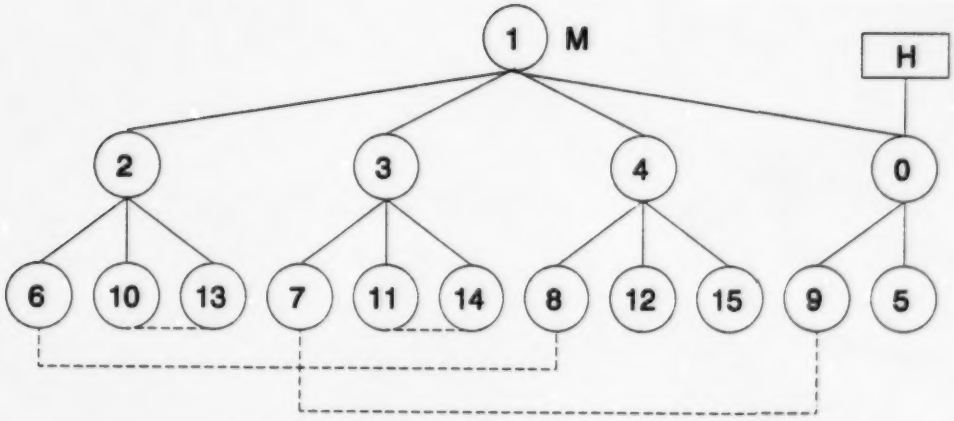


**Figure 4. "4–3" tree-topology**

The fifth experiment was dedicated to determining the efficiency of the parallel computation of the lower bound. The tests were conducted on the random problems, by using the PF implementation on four processors, and on the real application by using the EF implementation on 6 and 12 processors (connected by the 3–ary tree topology). Figure 5 displays the average sequential times (over three problems) for the random problems with the same number of products (AR1 to AR6). Table IV shows the average speedups (over three problems) for the same problems (AR2 to AR6) on four processors and the speedup for the real application (RA) on 6 and 12 processors. The first figure ("Global") represents the speedup computed on the total execution time, while the second one ("Lower bound") is the speedup obtained on the lower bounding subprogram. It is easily seen that the parallel implementation is quite efficient in solving the random problems but it is less so for the real application. This is due to the fact that, in this case, the networks for each product are of very different sizes, thus resulting in a poor balancing of the workload between processors. To overcome this problem,

we may design a load balancing strategy which takes into account the sizes of the networks. However, the identification and study of such a strategy is beyond the scope of this paper.

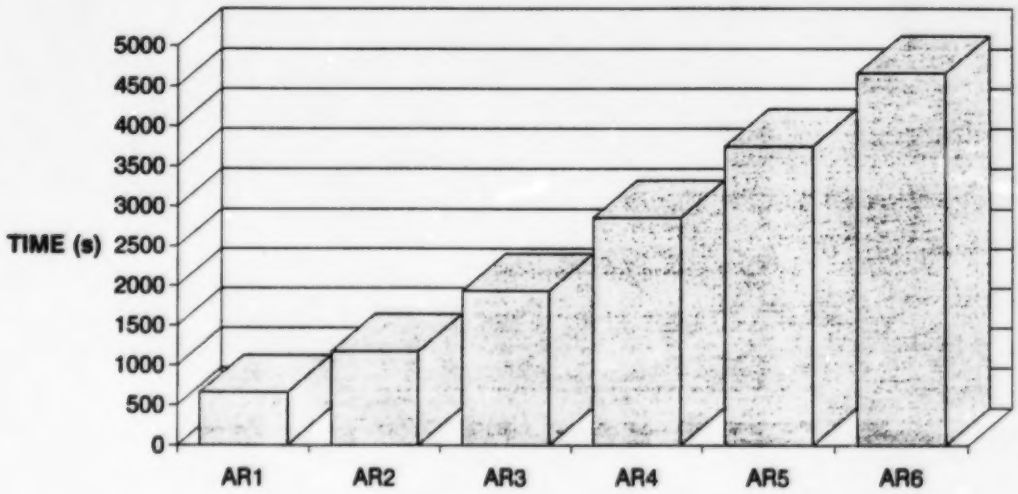


**Figure 5. Average sequential times for random problems**

**Table IV. Average speedups for the parallel computation of the lower bound**

| Problem | AR2 | AR3 | AR4 | AR5 | AR6 | RA | RA |
|---|---|---|---|---|---|---|---|
| *NPROC* | 4 | 4 | 4 | 4 | 4 | 6 | 12 |
| Global | 1.19 | 1.83 | 2.44 | 1.91 | 1.98 | 1.96 | 2.23 |
| Lower bound | 1.22 | 1.94 | 2.66 | 1.95 | 2.09 | 2.29 | 2.72 |

To summarize, we may now answer the questions stated at the beginning of the section:

1) It is both simple and robust to start the parallel exploration when the first feasible (integral) solution is reached. Though not necessarily "the best", this strategy ensures that, in most case, the parallel solutions obtained by a partial enumeration are at least as good as those obtained by the sequential algorithm, irrespective of the number of processors.
2) The depth selection criterion is efficient and completely outperforms the generation order criterion when the number of processors is high. This behavior is consistent to

the observation that, for a general branch-and-bound algorithm, feasible solutions are generally found in the deep portions of the search tree (Nemhauser and Wolsey, 1988). Thus, the depth criterion increases the chance of finding good solutions, more rapidly.

3) The speedup increases sublinearly with the number of processors.
4) In general, the speedups are better for problems with a larger number of products, but the performance depends on the time required by the sequential phase of the algorithm.
5) To implement a master-slave algorithm on a PC-based network of 16 Transputers, the 3–ary tree and the "4–3" tree are the best topologies if one wants to minimize communication time. However, the "4–3" tree increases the I/O time and cannot be easily generalized to a larger number of processors. Thus, the 3–ary tree is to be prefered.
6) The parallel computation of the lower bound is quite efficient when the networks for each product are of approximately equal sizes, while it is not so efficient in the opposite case.

Finally, note that we did not implement a combined algorithm, mentioned at the end of the previous section. To be useful, such a strategy would require a larger number of processors and more complex interconnection topologies.

## CONCLUSION

We have presented two parallel branch-and-bound algorithms for the multicommodity location problem with balancing requirements. The first algorithm implements a master-slave approach in order to perform a parallel exploration of the search tree. In the second algorithm, the multicommodity minimum cost network flow subproblem is decomposed by commodity in order to compute the lower bounds in parallel.

The computational experiments were performed on a PC-based network of Transputers. The results show that the first algorithm is most effectively implemented by combining a depth selection rule with a preliminary sequential phase, which may continue up to the identification of a first feasible solution. In this way, in most cases, the parallel solutions are at least as good as the sequential ones, irrespective of the number of processors. However, the algorithm suffers from three major drawbacks in terms of speedup performance: the length of the sequential phase, the synchronous aspect of communications, and the fact that it requires a lot of communications between the master and the slaves. These three factors explain the sublinear speedup that we observed.

The second algorithm is quite efficient. When the networks of the commodity subproblems are significantly different in size, the algorithm may be further improved by implementing a load balancing strategy which not only considers the number of products, but also takes into account the sizes of the networks for each product. However, even then, the level of parallelism to be achieved is limited by the fact that all applications have a very small number of products (in typical real-world applications, some twenty different container types are considered).

As a general conclusion, when, as in our experiments, few processors are available, the first algorithm is to be prefered. When the number of processors gets larger, a better strategy is to implement a combined algorithm, in which processors are organized in a hierarchy of

slaves. In a longer time frame, we have undertaken a more general study of the master-slave and collegial approaches to the parallel exploration of the branch-and-bound tree, in an attempt to overcome the drawbacks we have mentioned earlier.

## REFERENCES


Crainic,T.G., Dejax, P.J. and Delorme, L. (1989), "Models for Multimode Multicommodity Location Problems with Interdepot Balancing Requirements", *Annals of Operations Research* 18, 279–302.

Crainic, T.G., Delorme, L. and Dejax, P.J. (1989), "A Branch-and-Bound Method for Multicommodity Location with Balancing Requirements", publication #652, Centre de recherche sur les transports, Université de Montréal, forthcoming EJOR.

Gendron, B. (1991), "Implantations parallèles d'un algorithme de séparation et évaluation progressive pour résoudre le problème de localisation avec équilibrage", M.Sc. Thesis, Université de Montréal, publication #761, Centre de recherche sur les transports, Montréal.

Nemhauser, G.L. and Wolsey, L.A. (1988), "Integer and Combinatorial Optimization", *Wiley-Interscience*.

Microway (1990a), Quadputer Owner's Manual, Microway Inc.

Microway (1990b), Linkputer Owner's Manual, Microway Inc.

Parasoft (1990), Express FORTRAN, Parasoft Corporation.

Pardalos, P.M. and Rodgers, G.P. (1990), "Parallel Branch-and-Bound Algorithms for Quadratic Zero-One Program on the Hypercube Architecture", *Annals of Operations Research* 22, 271–292.

Quinn, M.J. (1990), "Analysis and Implementation of Branch-and-Bound Algorithms on a Hypercube Multicomputer", *IEEE Transactions on Computers* 39, 384–387.

Trienekens, H.W.J.M. (1986), "Parallel Branch-and-Bound on a MIMD System", Report 8640/A, Econometric Institute, Erasmus University, Rotterdam.

3L (1988), Parallel FORTRAN User Guide, 3L Ltd.